420-TP-013-001

Impact of Multi-Threaded Processes on the ECS Project

Technical paper - Not intended for formal review or Government approval.

June 1996

Prepared Under Contract NAS5-60000

RESPONSIBLE ENGINEER

Giuseppe Calavaro /s/ 6/26/96

Giuseppe Calavaro, Project Engineer Date
EOSDIS Core System Project

SUBMITTED BY

Arthur Palmer /s/ 6/26/96

Arthur Palmer, DE Manager Date
EOSDIS Core System Project

Hughes Information Technology Systems
Upper Marlboro, Maryland

This page intentionally left blank.

Abstract

This White Paper describes the potential impact that the use of multi-threaded processes may have on the ECS project. It identifies the current COTS that are not thread safe and recommends work around to allow their use. It also provides guidelines for consideration in future COTS procurements to avoid acquisition of products that are not thread safe. Finally, it provides guidelines to the ECS developers on how to write thread safe applications.

Keywords: Thread safe, multi-threaded processes, developer guidelines.

This page intentionally left blank.

Contents

1. Introduction

2. Thread Safe Issues

3. COTS Software Risk Assessment

4. Guidelines for Software Development

5. Conclusions

References

Abbreviations and Acronyms

1. Introduction

1.1 Purpose

The purpose of this White Paper is to provide an overview of the potential impact that the use of multi-threaded processes may have on the development and/or maintenance of the ECS.

This document has been written to:

1) help management in risk assessment and mitigation
2) identify non-thread safe COTS software used by this project that pose risk
3) recommend solutions and/or work around in using non-thread safe COTS software
4) provide guidelines to application developers for writing thread safe applications.

Reviewers of this document are expected to be the release system architects, the POCs for each COTS software product and the SEPG manager. Procurement Managers also have to be aware of this document and its impact in future software purchase.

1.2 Organization

This paper is organized as follows:

- Section 1 provides information about the white paper purpose, contents and whom to contact for more information.
- Section 2 provides an explanation of thread safe applications as related to multi-threaded processes.
- Section 3 provides the results from the thread safe risk assessment conducted on each of the COTS software products to be integrated in the ECS. It provides detailed discussions of COTS software considered to present potential risk and recommends final solutions and/or work around. This section also includes points of contact, vendors, and developers associated with each of the COTS software products.
- Section 4 provides guidelines for application developers to produce thread safe applications.

Questions regarding technical information contained within this Paper should be addressed to the following ECS and/or GSFC contacts:

- ECS Contacts
  - Giuseppe Calavaro, Project Engineer, (301) 925-1192, gcalavar@eos.hitc.com
  - Arthur Palmer, D. E. Manager, (301) 883-4012, ppalmer@eos.hitc.com

Questions concerning distribution or control of this document should be addressed to:

Data Management Office
The ECS Project Office
Hughes Information Technology Systems
1616 McCormick Drive
Upper Marlboro, MD 20774-5372

2. Thread Safe Issues

2.1 Problem Explanation

In modern operating systems, software processes can contain multiple threads of control, usually just called threads, or sometimes lightweight processes. In many respects, threads are like little mini-processes. Each thread runs strictly sequentially and has its own program counter and stack to keep track of where it is. Threads can create child threads and can block waiting for systems calls to complete, just like regular processes. Threads share the CPU just as processes do.

Different threads in a process are not quite as independent as different processes. All threads have exactly the same address space, which means they also share the same global variables. Since every thread can access every virtual address, one thread can read, write, or even completely wipe out another thread's stack. There is no protection between threads because (1) it is impossible, and (2) it should not be necessary. A process is always owned by a single user who has presumably created multiple threads so they can cooperate, not fight.

In addition to sharing an address space, all the threads share the same set of open files, child processes, timers and signals, etc. Given that there are these shared resources, a developer of an application, which will be instantiated in multiple threaded processes, must pay attention to insure that different threads are not competing for shared resources. That is, threads should not attempt to use the same resources at the same time. When facilities are employed in the development of applications to eliminate the risk of this competition between threads, the application is said to be “thread safe”.

ECS applications are going to use multiple threads in their processes. Therefore, special attention must be paid by the developers to insure that applications are thread safe.

ECS applications are going to use multiple threads in their processes. Therefore, special attention must be paid by developers to insure that applications are thread safe. When an application process spawns multiple threads and no actions have been undertaken to make sure that these threads are competing in a safe manner for the same resources (for example shared variables between threads), the application is very likely to crash. A crash may not occur under a controlled scaled development environment creating a false impression that the program is correct. However, considering the amount of time that an application will run, the number of different customer using an application, and the increased competition for resources that execution at the customer site presents, an application crash due to multi-threaded processes is almost certain to occur. Therefore, it is mandatory that every application, which will have multiple threaded processes, is tested in order to make sure that it is thread safe.

Additionally, most applications of this project are also integrated with COTS software. It is important to understand the ramifications and mitigate the risk associated with integrating non-thread safe COTS software with ECS applications. For example, in order to access Sybase, ECS applications use a library provided by Sybase itself. If this library is not thread safe, for instance

it uses some global variables, and two different threads of the same process use a function provided by this library there is the risk of an application crash.

2.2 General Resolution

Software applications can control access to shared data by defining critical regions which prevent multiple threads from accessing the same data at the same time. Critical regions are implemented using *mutex* or *condition variables*. The difference between mutexes and condition variables is that mutexes are used for short-term locking, mostly for guarding the entry to critical regions; condition variables are used for long-term waiting until the resources, contained in a critical region, become available. In generating applications that use multi-threaded processes, the developer must control the use of multiple threads so that they act in a safe way. Specific guidelines for developing thread safe applications are provided in Section 4.

Many COTS software products, such as database-management and GUI products, do not support concurrent or multi-threaded processes. They provide libraries, used by application developers, that are not be thread safe. A simple approach to writing thread safe applications is to consider the library a critical region. Therefore, a mutex could be locked before the call and unlocked upon return to eliminate competition for shared resources.

Another technique is to separate processes that are not thread-safe from those that are. This can be accomplished by having the server thread, for example, perform a fork and exec to create a new address space to execute the non-reentrant program. Communications between the new process and the server thread can be done using any inter-process-communications (IPC) facility available to the application.

The implementation in the IPC example suffers from the disadvantage that a separate process is created for each client request. This means additional use of system resources and degradation of performance with respect to the use of threads, which are much lighter than processes (i.e., less resource intensive). Section 3 provides specific problem resolutions for ECS COTS software which poses risk.

3. COTS Software Risk Assessment

3.1 COTS Software Risk Assessment Summary

In order to determine the risks associated with using COTS software products that do not support multiple threaded processes, all COTS software used on the ECS project, as of the date of this document, and that has an RFP[1] were assessed. Additionally, some COTS software, which did not undergo the RFP process, is also included for completeness. For example, RogueWave libraries do not have an RFP number. The COTS software products were categorized as COTS libraries, and development and test tools.

3.1.1 COTS Software Libraries

The thread safe problem with COTS software products generally arises when they provide libraries which the application developers call in their programs. If these libraries are not thread safe, risk is introduced to the application.

A relative indicator of overall risk was assigned to each product as follows: **High** identifies products which have and/or will cause problems; **Medium** identifies products which could cause problems; **Low** indicates that improper or incorrect use of a product could cause problems; and **None** identifies products that should cause no problems. A summary of the results of this assessment is presented in Table 3-1.

Table 3-1. Summary of COTS Software Libraries Risk Assessment (1 of 2)

RFP RFC	Product Description	COTS Name	Multi-Threaded Process Risk	Section Reference
#3	RDBMS for IMS server to support data query, location, and ordering of EOS and non-EOS data	Sybase	**High** - Sybase OpenClient 10 and DB-lib Sybase client library currently are not DCE thread safe. A new client library (XA-lib OpenClient), which supposedly fixes the problem is currently undergoing evaluation. That library is available for the HP and Sun platforms. Availability of a DCE thread safe XA-lib library for SGI is being negotiated (assuming that the current evaluations for the other platforms are favorable). However, because the RogueWave DBTools.h++ accesses the Sybase DB-lib, this XA-lib solution will not help those applications which use RogueWave DBTools.h++.	3.2.1
#5	Graphic User Interface development tool for Release A and throughout project development	BuilderXcessory Epak Widget Set GraphPak Widget	**Low** - BuilderXcessory: This tool generates code and must be evaluated to determine if this code is thread safe. **Medium** - Epak and GraphPak Widgets: These widgets are sub-classed and built from Motif widgets which are not thread safe.	3.2.2

[1] COTS software procurements are distinguished in the ECS project by Request For Procurements (RFP) numbers.

Table 3-1. Summary of COTS Software Libraries Risk Assessment (2 of 2)

RFP RFC	Product Description	COTS Name	Multi-Threaded Process Risk	Section Reference
#12	Math/Stat libraries to be installed at EDF, each DAAC, and SCF. Libraries. to support ANSI C, FORTRAN 77, FORTRAN 90.	IMSL	None	3.2.3
#14	Graphics and visualization SW to support image processing, map projections, correlation, filters, and contrast enhancement.	IDL	**Low** - Current version not thread safe. No problems have been reported on its use in a multi-threaded environment.	3.2.4
#17	Agent development kit for monitoring and control of remote management. applications; communicates status and control information between managed objects and management. applications.	Optima	None	3.2.5
#22	Backup Data Server DBMS to mitigate risk of scaling Sybase/SQS from Rel A to Rel B	Illustra Client Library	**Low** - Current version of Illustra is not thread safe, however, the way it is accessed in the system significantly reduces its risk. The next version 3.3 (Rel. B) will be thread safe.	3.2.6
#24	Integrated HTTP server, document search/indexing/ and document repository management. for Rel A	Netscape Server CGI	**Low -** Netscape: will be investigated in Rel.A, phase 3.	3.2.7
No bid: Sole source	C++ libraries	Tools.h++ DBTools.h++	None - Tools.h++ **High** - DBTools.h++: This product uses the Sybase client library DB-lib, which is not thread safe.	3.2.8 3.2.9
No bid: Sole source	General libraries for building X windows	Motif X11 R5	**Medium** - X11R5 is not thread safe and applications must insure single threads in use of X. Some facilities have been developed for the Server Request Framework (SRF) to mitigate this problem. X11Release 6 is expected to fix the problem.	3.2.10
RFI	Highly-open Enterprise Management Framework evolvable with new technologies and emerging standards.	OpenView	None	3.2.11
	Spatial Data Support for Sybase	SQS	None	3.2.12

3.1.2 COTS Development and Test Tools

Some COTS products are provided to support the development of application software. These products are typically code analyzers, test tools, load analysis tools, memory trace tools, and debuggers. While they are not a part of the final software product, they are instrumental in assisting developers in capturing problems early. Specifically, these tools were assessed for there support in detecting and mitigating potential thread safe problems with developer applications. A determination was made for each of these tools as to whether or not it provided support for evaluating thread safe code. A summary of the results of this assessment is presented in Table 3-2.

Table 3-2. Summary of COTS Development and Test Tools Risk Assessment

RFP RFC	Product Description	COTS name	Problem Supporting Thread Safe Code Development	Section Reference
#13	Capture/Playback test tool SW to conduct regression testing; identify max. system loads; and measure performance loads on developed SW for all project platforms.	LoadRunner/ XRunner	No	3.3.1
#19	Change Request Manager. Supports Nonconformance Reporting and Corrective Action (NCRA)	Purify Pure Coverage	No	3.3.2
	Compilers	SUN: SPARCompiler C++ 4.0.1 HP: C++ SoftBench License 3.50 DEC: DEC C++ 1.47 SGI: KAI C++ Compiler IBM: C Set ++	SUN: No HP: No DEC: **Yes**, C++ compiler does not have multi-thread process compile options SGI : No IBM: No	3.3.3
	Debuggers	SUN: iMPact 2.0 HP: xdb DEC: ladebug SGI: debugger IBM: dbx	SUN: No - Currently in the EDF to undergo trial use. HP: **Yes**, it does not have specific features to debug multiple threads. DEC: **Unknown**; being investigated SGI: **Unknown**; being investigated IBM: **Unknown**; being investigated	3.3.4

3.2 COTS Software Libraries Risk Assessment Detail

The assessment approach was to first, identify the COTS software that could potentially pose risk to the project. Secondly, evaluate and assess each of those COTS software products with input from the appropriate product expert. The POC for each COTS software product was asked to complete a form soliciting information about the potential risk that a particular COTS product may have on multi-threaded processes. COTS software which did not constitute any risk in this perspective did not undergo further investigation. Finally, a detailed analysis of the specific problems and resolutions involved with using each of these COTS software products in the ECS environment was performed. The information from the forms was synthesized with additional information received from other sources such as ECS developers, other people that were tracking such issues, product vendors, or external points of contact. The following sections address each of the products assessed.

3.2.1 Sybase Client Library

RFP number	3
Name of the product	Sybase
Name of the Company producing it	Sybase
Name of the Company that won the bid	Sybase
Vendor POC	Joe Shaffner (301) 896-1692 joseph.shaffner@sybase.com
Internal POC	EDS; Rel A: Bob Hartranft, x0997 2072G
COTS type	application and library
General description	Relational Database Management System

Multi-Threaded Process Risk

High - Sybase OpenClient 10 and DB-lib Sybase client library currently are not DCE thread safe. A new client library (XA-lib OpenClient), which supposedly fixes the problem is currently undergoing evaluation. That library is available for the HP and Sun platforms. Availability of a DCE thread safe XA-lib library for SGI is being negotiated (assuming that the current evaluations for the other platforms are favorable). However, because the RogueWave DBTools.h++ accesses the Sybase DB-lib, this XA-lib solution will not help those applications which use RogueWave DBTools.h++.

Problem Resolution

A work-around would be expensive. It involves converting the current client applications from multi-threading to multi-processing (at least for all database interface code). For Release A, it is too late to undertake such an effort.

A final solution rests in the successful evaluation of the latest version of the Sybase XA-lib client library and developing a retrofit and migration plan for its inclusion.

XA-lib testing status (April 1996): Sybase has published a new XA-lib OpenClient (EBF-5996). The Data Server test suite works without failing. The next step is to integrate the product into a Data Server ECS application; this will be accomplished during Release A phase III by DDS..

Product Dependencies

RogueWave Dbtools.h++ uses DB-lib only. Beta versions of DBTools that access Sybase using OpenClient areavailable but are not being used in ECS due to the risk involved with using a Beta release.

Technical Notes

Sybase support for ECS multi-threading applications is still under investigation. The most likely outcome is that a thread safe Sybase library (XAlib) will be made available for future use, DSS will verify that the library is indeed thread safe and compatible with DCE/OODC, which seems very likely.

In addition, the Sybase Open Client Release 11 client library (estimated availability Q2/96 for core platforms which will include Solaris), will also be thread safe and can be used with Sybase

Open Server R10. Open Client R11 can be used by those ECS developed applications which need multi-threading support without impact to the choice of server, other applications, or COTS software. Switching Release A to this library or not will depend on various factors (e.g., timing).

Sybase will offer XAlib with a set of Engineering Bug Fixes (EBF) which bring the XAlib up to the level of Sybase Open Client 10.0.3. The XAlib is available from Sybase and is supported by Sybase. The XAlib is available on HP, AIX, Solaris, but not on SGI. Sybase has no plans to port the library to SGI. DSS is determining the impact of this on the DSS operation. Currently, the SDSRV server for Release A is allocated to the SGI Challenger. It would need to move to a Sun workstation provided a suitable workstation is available, can take the load, and will not adversely affect the environment.

There are no known restrictions on how the XAlib can be used in comparison to CTlib. The library supports multi-threading through some serialization and through parallelised data structures, as appropriate. The current understanding is that it does allow truly concurrent interaction of multiple threads (using DCE threads) with Sybase.

There are no known conflicts between the XAlib and DCE. In particular, the library will not interfere with the operation of a DCE server. There are no poling or signaling conflicts. Sybase cautioned us against trying to use open server components with DCE (but we do not have a corresponding requirements). Based on this information, it was concluded using XAlib now (for development and testing) and switching to Open Client R11 later in the year would be the best approach. DSS is finding out whether our contract covers XAlib and R11 client upgrades, costs, and availability of a trial versions.

DSS is also developing a plan for verifying that XAlib indeed fixes the problem. It was decided that tests should consist of putting up an OODCE server listening for requests, and firing up multiple threads interacting with Sybase. This will test both potential conflict with DCE/OODCE and multithreading. The plan should be based on the time frames for getting XAlib, etc.

There exists an underlying concern even with a transition to XAlib and Open Client Release 11, in that ECS applications use the RogueWave DBTools.h++ libraries which require linking with DBlib. DBlib is not thread safe, and it currently cannot be assumed that a thread safe version is forthcoming. CSS Release A is still investigating whether DBtools protects the database client library from being multi-threaded by itself, or whether applications need to do that. Until this is clear, applications cannot assume that DBtools' use of DBlib is thread safe.

Additionally, RogueWave cannot use XAlib, and programs which require multi-threaded access to Sybase cannot use RogueWave to build their interface. This is because RogueWave uses DBlib, whereas XAlib is an incarnation of a CTlib, which is incompatible with the DBlib function calls. (There was some speculation as to the future availability of a CTlib compatible version of RogueWave, but it was concluded that depending on a RogueWave upgrade for Release A would not be a good idea).

Design Rule

As a result, applications which require multi-threaded connections to Sybase must be linked with the appropriate version of XAlib and may not make use of DBtools. Applications which can use single-threaded Sybase connections may use DBtools, however, until further notice, developers must ensure that the database client is used by only one thread at a time and locking is used.

Based on planning (e.g., time needed for verifying XAlib) and the results of the XAlib evaluation, a decision must be made as to whether the need to pursue the multi-process option as a fallback is feasible. For the moment, this will be put on hold. Finally, it must be determined which ECS subsystem applications have a need for multi-threaded interaction with Sybase.

3.2.2 BuilderXcessory, Epak and GraphPak Widget Set

RFP number	5
Name of the product	BuilderXcessory Epak Widget Set GraphPak Widget Set
Name of the Company producing it	Integrated Computer Solutions, Inc.
Name of the Company that won the bid	Integrated Computer Solutions, Inc./ McBride
Vendor POC	Voice: (617) 621-0060, Fax: (617) 621-9555 E-mail: (general info) info@ics.com (Tech support) support@ics.som (License Keys) keys@ics.com
Internal POC	ECS, Rel.A, Randy Wilke x0818 2113E
COTS type	These are applications that produce code integrated to our custom code.
General description	**BuilderXcessory** is a GUI builder application that a developer uses to graphically define a user interface. A feature used by ECS developers is to generate C++ code. **Epak** and **GraphPak** are Motif based widget sets that can be used stand alone or in combination with BuilderXcessory. Epak is general purpose widgets. GraphPak is widgets for creating graphs of various kinds.

Multi-threaded process risk

Low - BuilderXcessory is simply a development tool and does not need to be thread safe. However, it is being used to generate code for the ECS project. Whether all of the generated code is thread-safe is to be determined.

Medium - Epak and GraphPak, the actual ICS-written libraries, which are independent from Motif calls, must undergo further analysis to determine if they are thread safe, but they are not thread-safe with X11R5.

Problem Resolution

Source code licenses for these products are available (FOS group might have one). Presumably, unsafe code could be made safe. Gauging the effort to accomplish this requires more investigation.

Product Dependencies

BuilderXcessory makes calls to Motif libraries, which are known to not be thread safe. Also, the Epak and GraphPak widgets, while written by ICS, are sub-classed off of Motif widgets and built using the standard Motif APIs. There is potential for risk as a result of this.

3.2.3 IMSL

RFP number	12
Name of the product	IMSL
Name of the Company producing it	Visual Numerics
Name of the Company that won the bid	Visual Numerics
Vendor POC	Anthony Colyandro, anthony@houston.vni.com
Internal POC	Larry Klein x0764 rm.2022
COTS type	library
General description	Math/Stat libraries to be installed at EDF, each DAAC, and SCF. Librs. to support ANSI C, FORTRAN 77, FORTRAN90.

Multi-Threaded Process Risk

None - IMSL has a multi-thread version.

Problem Resolution

N/A

Product Dependencies

None

3.2.4 IDL

RFP number	14
Name of the product	Interactive Data Language (IDL)
Name of the Company producing it	Research System Inc.
Name of the Company that won the bid	Research System Inc.
Vendor POC	Kevin Jackson, (301)595-3790, kevinj@rsinc.com David Uhlir, (303) 413-3904, uhlir@rsinc.com
Internal POC	Brand Fortner x0779 rm.2079
COTS type	application and a library
General description	Interactive Data Language (IDL) is a set of tools for graphics and visualization to support image processing, map projections, correlation, filters, and contrast enhancement. It is a programming language with a complete graphic interface. It is also possible to call by other applications.

Multi-Threaded Process Risk

Low - IDL is not multi-threaded, however, Research System Inc. says "In many ways, IDL's single-threadedness makes it a less-problematic, lower-risk component of a multi-threaded environment than the multi-threaded applications themselves."

Problem Resolution

N/A

Product Dependencies

None

Technical Notes

Research System Inc. says:

"We are unaware of any special work-around necessary to make IDL 'safe' in a multi-threaded environment - assuming the multi-threaded applications (not IDL) in that environment have sophisticated process management capabilities to keep from stepping on each other. When an IDL session is going, its (single) process will be evident to the operating system and will be more stable (in terms of memory areas) than any multi-threaded applications running at the time. In other words, if other single-threaded applications are okay to run in the multi-threaded environment, IDL will run fine, and will not interfere with the other processes. That multi-threaded processes do not interfere with IDL rests completely on the sophistication of the multi-threaded environment. If a multi-threaded application is crude enough to write into IDL's memory space, over the objections of the operating system, then bad things are likely to happen. This kind of behavior would make the multi-threaded application quite unstable all by itself - regardless of whether IDL is present or not."

3.2.5 PEER Optima

RFP number	17
Name of the product	Optima
Name of the Company producing it	Peer
Name of the Company that won the bid	Peer
Vendor POC	Lisa Radding (408)-556-0720
Internal POC	Chris Kingsbury x0544 rm.2109C
COTS type	development kit consisting of source code.
General description	SNMP development kit. Simplifies the integration of the SNMP into C/C++ based applications. Agent for monitoring and control of remote mgmt. applications; communicates status and control info between managed objects and mgmt. applications.

Multi-Threaded Process Risk

None

Problem Resolution

N/A

Product Dependencies

None

3.2.6 Illustra Client Library

RFP number	22
Name of the product	Illustra
Name of the Company producing it	Illustra Information Technologies
Name of the Company that won the bid	Illustra Information Technologies
Vendor POC	Jackie McAlexander (301) 214-9064
Internal POC	Wayne Lewis x0737 rm.2081C
COTS type	application
General description	Backup Data Server DBMS.

Multi-Threaded Process Risk

Low - Connections to the database are made through the Elan License Manager which is not thread safe. This library will be thread-safe in its next version 3.3 of the product (Rel.B).

Problem Resolution

There does not appear to be a work around for version 3.2, but may not be an issue because of the way it will be used.

Product Dependencies

None

<{To Be investigated:

Illustra Data Blade API.

Moreover, consider server back-end API data blade interface.

3.2.7 Netscape Server CGI

RFP number	24
Name of the product	Netscape
Name of the Company producing it	Netscape
Name of the Company that won the bid	Illustra Information Technologies
Vendor POC	Different for each product. Prime contractor is Illustra. Contact: Therasa Aschenbrenner x0841 rm 1054A
Internal POC	Graham Vowles x0586 rm.2038J
COTS type	application and a library
General description	These are three products that Illustra put together to win the bid fir the RFP 24. Netscape is the http server, Topic is the search and indexing tool and Illustra is the ORDBMS document mgmt.

Multi-Threaded Process Risk

Low - Netscape: Will be investigated in Release.A phase 3.

Problem Resolution

There does not appear to be a work around for version 3.2, but may not be an issue because of the way it will be used.

Product Dependencies

None

3.2.8 Tools.h++

RFP number	No bid. RogueWave is the sole producer.
Name of the product	Tools.h++
Name of the Company producing it	RogueWave
Name of the Company that won the bid	N/A
Vendor POC	Dennis Kennedy, (503) 754-2311, kennedy@roguewave.com
Internal POC	EDS Rel A: Dmitri Schoeman x0852 rm.2111C
COTS type	library
General description	Provides a large number of utility classes for building C++ applications.

Multi-Threaded Process Risk

NoneMedium - The library is now thread safe to the extent that your operating system supports thread-safe system calls. Where possible, it also makes only thread safe system calls.

Problem Resolution

N/A

do not**Product Dependencies**

None

Technical Notes

Extract from the Release Notes for Tools.h++ Version 6.1:

We have added multi-thread support for environments that support POSIX threads. However, we have not yet automated the detection of such environments. If you'd like to build a multi-thread safe version of the library, and your environment supports POSIX threads (via the header file "`pthread.h`"), add the line:

```
#define RW_POSIX_THREADS
```

to the Rogue Wave header file "`rw/compiler.h`". Uncomment the appropriate lines in the toolsrc/makefile as per the instructions there, and rebuild the library.

The only non-POSIX platform that Rogue Wave supports for threads is Sun Solaris. They did say that the mapping of their threads calls to another non-POSIX environment wouldn't be too difficult for someone, since they "do not do anything too exotic".

The Tools.h++ library will work in the presence of multiple threads as it does in a single thread environment provided that you either do not share objects between threads, or lock between accesses to objects that are shared between threads. Internally, the library has enough locking to maintain its own integrity.

The `setlocale()` operations are not thread safe anywhere, so creating `RWLocaleSnapshot` instances is not either. However, instances created when no other thread is running may safely be used by multiple threads.

Supposedly Rogue Wave is working on a version which will allow multiple threads to concurrently reference an object, but we do not have enough information about it at this time.

Tools.h++ libraries were believed to be not thread safe on SGI. The problem was that the libraries were not built using the Posix thread flag. CSS Release A rebuilt the libraries for SGI and now Tools.h++ is thread safe on SGI too..

3.2.9 DBTools.h++

RFP number	No bid. RogueWave is the sole producer.
Name of the product	DBTools.h++
Name of the Company producing it	RogueWave
Name of the Company that won the bid	N/A
Vendor POC	Hongyan Li (541) 754-2311, Li@roguewave.com or support@roguewave.com
Internal POC	EDS; Rel.A: Quan Luu x0386 rm.2073D
COTS type	library
General description	Provides a simple mechanism to interface C++ programs with relational DBMS; Release A subsystems use it for interfacing with **Sybase**.

Multi-Threaded Process Risk

High - The product currently does not support the multi-threaded version of the Sybase client.

Problem Resolution

One possible work around to mitigate risk with using this product is to apply mutexes whenever an OODCE server accesses Sybase. Sybase's underlying proprietary API, "`db-lib`", is not thread safe. RogueWave DBTools.h++ is built on top of this API. When building an OODCE Server, you should ensure that all accesses to Sybase are guarded using a single `DCEPthreadMutex` object. This is an obvious performance hit, but nothing can be done about this until Sybase provides an underlying API that is thread-safe and it is also supported by DBTools.h++.

Another known problem is in the way that RogueWave DBTools.h++ manages its connections to Sybase. You should ensure that for a given `RWDBDatabase`, you maintain one `RWDBConnection`. If you do not, your server will hang when trying to access Sybase after just a few RPC's from an OODCE Client.

A beta copy of RogueWave DBTools.h++ version 1.1.1 and the Beta access libraries for CT-LIB is being evaluated for future use to resolve the multi-threaded process related problems encountered with the Release A MSS (see Technical Notes below). This version can be interfaced with the thread safe version of the Sybase client. If this works, then we will able to use the XA library version of CT-LIB to build all of our servers on Sun, HP, and SGI (assuming

a successful port to SGI by the vendor will happen). Additionally, verification must be made of availability of this product for the DEC platform.

Product dependencies

Sybase

Technical Notes

Supporting details on the problems encountered with the Release A MSS are provided below:

MSS has written an OODCE server to manage user profile information in Sybase. A client may request various operations on user profiles via this server (insert, update, delete, retrieve, etc.). The server uses RogueWave DBTools.h++ to access Sybase. This server was running on the HP platform (Cyclops).

We discovered that the first few times a client connects to the server, everything works fine. However, after just a few calls, the OODCE server hangs indefinitely while trying to access Sybase.

You need to ensure that your OODCE server maintain a single `RWDBConnection` to Sybase and that it passes this object to all "`RW`" DBTools.h++ methods that accept an `RWDBConnection` object as a parameter. Otherwise, you will get the strange hang-up behavior just described.

Our server creates a global profile Database Object that contains and initializes a `RWDBDatabase` object when it first comes up. Call this data member `db`. When clients make requests and the server needs to access Sybase, we always provide the `RWDBConnection` object to those `RW` DBtools.h++ methods that accept one.

The following code fragments are simplifications of the MSS code. We actually maintain a global Profile Database Object using the Singleton Design Pattern, which is a standard technique for creating and accessing global C++ objects. See "Design Patterns", by Addison Wesley.

```
// In source file for OODCE server:
RWDBDatabase db;
DCEPthreadMutex sybaseLock;

int main()
{
  db = RWDBManager::database("SYBASE",
    "cyclops_srvr",
    "username",
    "userpassword",
    "mss_db");
  // rest of OODCE initializations
}

// In source file where OODCE object methods are kept:

void DistributedObject::GetProfile()
{
```

```
  sybaseLock.Lock();
  RWDBTable mytable = db.table("MsAcUsrProfile");
  RWDBSelector selector = db.selector();
  selector << mytable["usrName"] << mytable["homeDAAC"];
  RWDBReader reader = selector.reader(db.connection());  // HERE IT IS!!
  // etc
  sybaseLock.Unlock();
}
```

This is still a troublesome problem. One would have hoped that DBTools.h++ would have detected a problem and thrown an exception. It's also unclear what to do if an applications needs several connections open at once.

3.2.10 X R5/Motif

RFP number	No bid
Name of the product	X R5/Motif
Name of the Company producing it	Hardware vendors
Name of the Company that won the bid	N/A
Vendor POC	N/A
Internal POC	HTSC Help Desk, x0909
COTS type	library
General description	General library for building X windows/Motif GUI interfaces.

Multi-Threaded Process Risk

Medium - As of X11, Release 5, X-based programs are not thread safe. Applications must use single threads in their X use. This presents a problem for Rel A client applications which call servers using OODCE, since asynchronous call backs may be involved. X does not present a significant problem in its use to support GUI development.

Problem Resolution

Release 6 of X is expected to fix the multi-threaded problem.

Steps can be taken to let multiple control flows exist in a process that is not thread safe. You can do this by making sure that only one thread, usually the main thread, performs all the X-related processing and that the remaining threads, using IPC pipes mechanism, communicates to the X event loop manager as if they were separate processes.

Therefore, all X-related operations are performed as part of the main thread and the RPC call is issued from a separate thread. The RPC thread and the main thread are connected with a unidirectional IPC pipe. The read end of the pipe is connected to the event loop manager using the `XtAddInput` function call and the write end of the pipe is recognized by the RPC thread because it is a global variable. When the RPC is issued, the thread sending the call will block and the main thread can continue to accept user input. When the RPC completes, a string of characters is written to the pipe. The writing of this data will cause the X-event loop manager to

call the call-back routine specified during the `XtAddInput` function call. This call-back routine performs the application operations on the main thread in a thread safe fashion.

A thread safe version of the Server Request Framework (SRF) has been designed and is slated for implementation in Rel A, Phase 3. It will deal with asynchronous client/server requests and subscription notifications such that they can be used safely by X applications. The technical approach will be communicated to the SDSRV development team so that they can provide a similar solution. Developers who believe that their X Applications have multi-threading requirements beyond asynchronous requests/subscription notifications must identify these requirements to the appropriate Release architect immediately.

Clients not using SRF must employ work around, which may involve use of multi-processing (i.e., run the X application in one process, and the client API in another).

Product dependencies

None

3.2.11 HP Openview

RFP number	No RFP found
Name of the product	Openview
Name of the Company producing it	HP
Name of the Company that won the bid	N/A
Vendor POC	HP Help Desk. 1-800-633-3600 id=ECS/EDF
Internal POC	Scott Austin x1143 rm.2080F
COTS type	application
General description	Highly-open Enterprise Management Framework evolvable with new technologies and emerging standards.

Multi-Threaded Process Risk

None

Problem Resolution

N/A

Product Dependencies

None

3.2.12 Spatial Query Server (SQS)

RFP number	No RFP found
Name of the product	Spatial Query Server (SQS)
Name of the Company producing it	Vision International
Name of the Company that won the bid	N/A
Vendor POC	David Wiseman, (703) 658-6123, dwiseman@autometric.com
Internal POC	Bob Hartranft x0997 rm.2072G
COTS type	application
General description	High performance open server application that allows the user to include geographic constructs using Sybase SQL server.

Multi-Threaded Process Risk

None - SQS does not create any multithreading problems for the DataServer. SQS is not a DCE application and thus does not use DCE threads. It is a Sybase Open Server Application. The DataServer will communicate with SQS using a DCE thread safe version of Open Client.

Problem Resolution

N/A

Product Dependencies

None

3.3 COTS Development and Test Tools Risk Assessment Detail

The development and test tools assessment approach followed the same as that described for the COTS software libraries. The following sections address each of the products assessed.

3.3.1 LoadRunner/XRunner

RFP number	13
Name of the product	LoadRunner XRunner
Name of the Company producing it	Mercury
Name of the Company that won the bid	Mercury
Vendor POC	Mercury Customer Support (408) 523-4299
Internal POC	Tom Collins x0441 rm.3113B
COTS type	application and library
General description	Capture/Playback test tool SW to conduct regression testing; identify max. system loads; and measure performance loads on developed SW for all project platforms.

Problem Supporting Thread Safe Code Development

No - The customer support staff has not heard of or come across any problems using XRunner/LoadRunner in a multi-threaded environment.

Problem Resolution

N/A

Product Dependencies

None

3.3.2 Purify and PureCoverage

RFP number	19
Name of the product	Purify PureCoverage
Name of the Company producing it	Pure Software
Name of the Company that won the bid	Pure Software
Vendor POC	
Internal POC	Tom Suhrstedt x0401 rm.2108A Jim Ruckstuhl x0394 rm.3109B
COTS type	application
General description	**Purify**: It is a run-time memory error detection tool. Purify presents the code that caused memory leaks in an easy to use interface. **PureCoverage**: It is a code coverage improvement product designed to be used by developers during daily unit tests. Developers can use an annotated source view that provides line-by-line analysis of either tested or untested code

Problem Supporting Thread Safe Code Development

No - This product supports debugging of multiple threaded processes.

Problem Resolution

N/A

Product Dependencies

None

3.3.3 Compilers

RFP number	No RFP found
Name of the product	SUN: SPARCompiler C++ 4.0.1 HP: C++ SoftBench License 3.50 DEC: DEC C++ 1.47 SGI: KAI C++ Compiler IBM: C Set ++
Name of the Company producing it	
Name of the Company that won the bid	N/A
Vendor POC	
Internal POC	HTSC Help Desk
COTS type	application
General description	C++ compilers that are in the software development environment baseline for Release A as multi-threaded debuggers.

Problem Supporting Thread Safe Code Development

Yes - The DEC C++ compiler does not support its "-thread_safe" option, which allows the creation of thread safe code.

Problem Resolution

A decision must be made to use another C++ compiler on the DEC platform or wait until DEC will support its thread safe option. No one has been assigned to perform the task to evaluate this yet.

Product Dependencies

N/A

Technical Notes

Compilers are provided by the platform vendors. Every compiler has some option that should be used when compiling programs that have multiple threads in the same process. SUN, HP, SGI and IBM have these options which work correctly. The DEC C++ compiler does not support its "-thread_safe" option, which allows the creation of thread safe code. Therefore, at the moment, it seems that it is not possible to compile multi-threaded applications on this platform.

3.3.4 Debuggers

RFP number	No RFP found
Name of the product	SUN: iMPact 2.0 HP: xdb DEC: ladebug SGI: debugger IBM: dbx
Name of the Company producing it	
Name of the Company that won the bid	N/A
Vendor POC	
Internal POC	HTSC Help Desk
COTS type	application
General description	Debuggers that are in the software development environment baseline for Release A as multi-threaded debuggers.

Problem Supporting Thread Safe Code Development

Yes - Debuggers are generally provided by the platform vendors as part of the bundled compiler software. These vendors assert that their single thread debuggers can handle multi-threaded applications. This should be interpreted to mean that their debugger can execute against multi-threaded applications without problem, but it does not provide any special mechanisms to select, monitor, or trace a specific thread during program execution. The only exception is the SUN debugger iMPact, which provides mechanisms to detect and trap activities occurring on specific threads. Without such a facility, it is difficult and sometimes a cumbersome task for developers to determine with any certainty what thread of program execution is being viewed.

Problem Resolution

There is no simplistic work around that will satisfy all of the various platform configurations.

Product Dependencies

N/A

Technical Notes

The SUN debugger, iMPact, is the only one that provides the ability to trace a single thread. All the others, (i.e., HP, SGI, DEC, and IBM) although can execute against multi-threaded applications without problem, do not provide capabilities for examining, activating, or breaking on threads. Without such facilities, it is difficult and sometimes a cumbersome task for developers to determine with any certainty what thread of program execution is being viewed.

3.4 Guidelines for Future COTS Software Procurements

Every COTS software purchase that includes client libraries, therefore, all software that will be included in our ECS developed executables, must fulfill the requirement that it is thread safe.

This requirement should be written case by case in the RFP by an engineer aware of thread safe problems and the use of the specific COTS.

4. Guidelines for Software Development

Threads allow parallelism to be combined with sequential execution and blocking system calls. Blocking system calls make programming easier and parallelism improves performance. They are very useful both in the case of multi-processor or single-processor architectures allowing easier application design, code writing and maintenance.

For example, one thread, the dispatcher, reads incoming requests for work from a system mailbox. After examining the request, it chooses an idle (i.e. blocked) worker thread and hands it the request, most likely by writing a pointer to the message into a special word associated with each thread, the dispatcher then wakes up the sleeping worker (e.g., by signaling the semaphore on which it is sleeping).

Threads are frequently also useful for clients. For example, if a client wants a file to be replicated on multiple servers, it can have one thread talk to each server.

Since threads share a common memory, they can, and usually do, use it for holding data that are shared among multiple threads. Access to shared data is usually programmed using critical regions, to prevent multiple threads from trying to access the same data at the same time.

Threads are a part of DCE: the **Pthreads** package is included as part of DCE, and RPCs also use threads. Since the ECS developers will be using multi-threaded processes, and will be providing code for use in a multi-threaded environment, they need to understand and use threads correctly.

The following sections provide guidelines for programming with threads. This information is intended to help ECS application developers get started using threads. For a more complete understanding of this topic the reader should refer to Operating Systems [Tanenbaum92] or DCE books such [Shirley92].

The information is organized as follows:

- When To Use Threads
- Starting And Ending Threads
- Thread safe Function Calls
- Using Non-thread Safe Libraries
- Warnings About Non-thread Safe Function Calls
- Atomic Operations
- Mutexes
- Exception Handling
- Using Multi-thread Options In Makefiles
- The Real World
- Suggestions And Reminders

4.1 When To Use Threads

The reasons to use multiple threads in a process have been already presented. As a quick reference and general guideline application developers should consider using threads for:

- parallel activity
  - if two activities can occur concurrently
  - if connected to other machines over a network and it is desired to have each connection handled separately
- periodic activity
  - if an activity occurs periodically that responsibility can be delegated to a single thread and have that thread sleep until it is needed again
- conceptually distinct components
  - if a specialized routine is required to be autonomous (for example, a garbage collection routine).

4.2 Starting And Ending Threads

Creating Threads

Threads can be created with a single function call in C++:

```
DCEPthread my_thread(&attr, function, argument);
```

(Note: the `pthread_create()` function can be used in C).

Thread Arguments

A thread will only accept a single argument. If you wish to pass multiple variables to a new thread you should place them in an object or structure and pass a pointer to the structure. No data is copied. When you pass a pointer to a structure into a thread, it uses the same copy of the data that the parent thread uses. It is important to remember this before you change or delete any of that data. The starting function for a thread must be cast as type `DCEPthreadProc` and the argument as `DCEPthreadParam`.

When joining a thread, the type returned will be of type `DCEPthreadParam`. You will probably need to cast it to the appropriate function return type before being able to manipulate it.

Deleting Threads

The return status from a thread is determined by the return value of the function passed to the thread start method. If you want the return value of the spawned thread, call the member function `join()`. (Note: the pthread_join() function can be used in C)

If you do not want the return value, but want to wait until the thread is completed before exiting the local scope, then set the "`attr`" variable used above to `join_on_delete`. This means that

when the destructor for the DCEPthread object is called, before the call returns it will wait until the thread has finished executing.

```
DCEPthreadAttr attr;
attr.Termination(Pthread_join_on_delete);
```

Multiple Threads Example

The following code segment multiplies two pairs of numbers, each pair being multiplied as a separate threaded process, and then adds the products from these separate threads to produce a result.

```
#include <oodce/Pthread.H> // Always place first
#include <iostream.h>

typedef struct _input_t
{
    int val1;
    int val2;
} input_t; // Create a single type that can hold 2 ints

int multiply(input_t *input)
{
    return(input->val1*input->val2); // multiply
}

main()
{
    input_t pair1, pair2; // create two pairs of ints
    pair1.val1=1; pair1.val2=2; // initialize them
    pair2.val1=3; pair2.val2=4;

    DCEPthreadAttr attr; // define an attribute (unused)
  // Now create and start the two multiplies
    DCEPthread DCEPthread1(&attr, (DCEPthreadProc)
     multiply,(DCEPthreadParam) &pair1);
    DCEPthread DCEPthread2(&attr, (DCEPthreadProc)
     multiply,(DCEPthreadParam) &pair2);

  // Get the results of Join() (return values)
    int result = (int) DCEPthread1.Join()
         + (int) DCEPthread2.Join();

    cout << "result = " << result << endl;
}
```

4.3 Thread Safe Function Calls

Unfortunately, many operating systems were not originally designed to be multi-threaded (including early standard versions of UNIX). Some function calls maintain state between calls and do not work in a multi-threaded environments. Thus, not all libraries supplied with an

operating system version are thread safe. Often, non-thread safe functions have thread safe replacements supplied. For example, if you look at the "man page" for each of the following functions of the form `<function>_r` (e.g., `ctime_r`) you will see that a thread safe version for each exist: `ctime, localtime, asctime, gmtime, ctermid, rand, getlogin, readdir, strtok, tmpnam`.

4.4 Using Non-thread Safe Libraries

Unfortunately, you cannot always use thread safe libraries because many times they are simply not supplied. If you must use a library that is known not to be thread safe you must take precautions. One thing you must do is surround non-thread safe code with calls to `pthread_lock_global_np()` and `pthread_unlock_global_np()`. This will prevent conflict with thread safe system calls. It will also only allow a single thread at a time to be executing any code between those function calls.

You also must be careful to consider what will happen if multiple threads take turns accessing the non-thread safe libraries. For example, consider the possibility that the library may save state information from one thread that may not be valid for another thread.

4.5 Warnings About Non-thread Safe Function Calls

Code may fail in unexpected places in unexpected ways as a result of use of non-thread safe functions. For example, consider a scenario where you do not surround non-thread safe code with `pthread_lock_global_np()` and `pthread_unlock_global_np()`as mentioned earlier. One thread begins to use a thread safe `malloc`. It determines what memory address it will allocate. Immediately after that a non-thread safe function calls a non-thread safe `malloc` and allocates that same block of memory. Now the first thread continues executing using memory it thought was free but it is actually in use by the second thread.

Non-thread safe function calls may cause other threads to act improperly. For example, on the HP, if a non-thread safe library calls `sleep()` the entire process will be blocked for the duration of the sleep period, not just the calling thread. Blocking system calls in general may block an entire process. For example, network or disk reads or writes on the HP would block the entire process, not just the calling thread.

4.6 Atomic Operations

An atomic operation is an operation that can not be interrupted. Once an atomic operation starts it will execute without interference from any other operations. On some hardware platforms there is an machine level instruction called '`test and set`' which will test that a memory location has a specific value, and if so set it to another value. This operation can not be interrupted. On many platforms this instruction is the basis for multi-threaded interactions.

There are no atomic operators that C or C++ programmers can access. Even the simple task of reading a variable is not atomic -- it may be interrupted. Therefore, any operations that are

executed as multi-threads are at risk of being prematurely interrupted or interfered with if a thread safe mechanism is not employed.

4.7 Mutexes

A mutex is a MUTual EXclusion lock. It is a tool, which when used correctly, allows programmers to make code thread safe. It is a lock to allow threads serial access to a shared resources, much like in C `lockf()` will allow serial access to a file.

Types of Mutexes

There are three basic types of mutexes fast, recursive and non-recursive:

With fast mutexes, the lock may only be locked once before being unlocked. If the same thread tries to lock it a second time, while already holding the lock, the thread will wait forever (since it can't unlock it.) The fast mutex is the default mutex type.

The lock of recursive mutexes may be locked by the same thread any number of times. However, it also must unlock the lock that number of times. This is often useful inside recursive functions.

Finally, the lock of non-recursive mutexes may only be locked once before being unlocked, like the fast mutex. However, if a thread attempts to lock the lock when it already holds the lock, it will return an error.

Applying Mutexes

Application developers should consider using mutexs when:

- two threads share memory
- two threads share some exclusive device
- any static data shared between threads needs to be protected
- multiple threads are allowed to reference the same object and the member data must be protected
- sending data to common files, or to cout, to prevent interleaving of data to each of those output devices.

In using a mutex, a developer should lock a mutex before accessing a data member and unlock it after accessing it. If you normally modify and read several variables together, it is common to use a single mutex to lock all of them. When locking a mutex, if the mutex is already locked by another thread, your thread will wait until the mutex is unlocked before it locks it and returns. The following code segment illustrates the application of mutexes:

```
#include <oodce/Pthread.H>
#include <iostream.h>

class myclass
{
```

```
private: // internal data members
    static DCEPthreadMutex counter_lock;
    static int counter; // protected by counter_lock
public:
    myclass();
    ~myclass();
    static int how_many_instances();
};

// The following are needed for the static variables
DCEPthreadMutex myclass::counter_lock;
int myclass::counter;
// Have the class keep track of how many class objects
// are around. Use counter_lock to protect the counter.

myclass::myclass()
{
    counter_lock.Lock(); // Lock the lock
    counter++; // Since I hold the lock, I can modify
    counter_lock.Unlock(); // Unlock the lock
}

myclass::~myclass()
{
    counter_lock.Lock(); // Don't forget to lock
    counter--;
    counter_lock.Unlock(); // And unlock
}

int myclass::how_many_instances()
{
  // ...
}

// Function to read how many instances of the class exist
// Remember to lock the mutex before reading and unlock
// after reading. Mutexes only protect data if you
// remember to lock them.

int myclass::how_many_instances()
{
    counter_lock.Lock(); // Lock the mutex
    int retval = counter; // copy the data
    counter_lock.Unlock(); // unlock the mutex
    return retval; // return the *copy*
}
```

(Note: by the time the calling function does anything with the value from `how_many_instances()` it may have changed, however, it will be guaranteed to be a valid value and correct at a point during the function call).

Additional Programming Aids

Additional functions that can be used in constructing thread safe applications are listed below:

- `TryLock` (or `pthread_mutex_trylock` - This function will attempt to lock the lock However, if another thread already holds the lock, rather than waiting for that thread to relinquish the lock, it returns failure.
- `pthread_lock_global_np()` and `pthread_unlock_global_np()`- These function calls lock and unlock a mutex shared by the entire application. These were mentioned before for use with non-thread safe code.
- `pthread_once` - This function will guarantee that only one call to the specified function is ever used. This is useful for one time initialization.

Notes and Warnings

- There are no C or C++ atomic operations. This includes reading a variable, and setting a variable. You must use mutexes to protect any data that may be concurrently read from or written to.
- Mutexes are only conventions for restricting access to variables. If you do not use them properly your code will not work properly.
- When using mutexes, remember that they are shared resources. Try to keep them locked for as short a time as possible.
- Remember that you need to lock a mutex before reading or writing shared resources.
- When a thread must lock multiple mutexes it is important that the mutexes are always locked in the same order to prevent deadlock.
- Use set and get member functions where possible to access member variables. This will help insure that mutexes are used properly whenever accessing the data.
- Which ever way you chose to make your code thread safe, clearly document if it is okay for multiple threads to access the same object.

4.8 Exception Handling

Although C++ exception handling was designed to work in conjunction with multi-threaded applications, thrown exceptions might generate unexpected side effects in multi-threaded applications. The basic problem with exception handling in multi-threaded applications is that C++ exceptions are processed on the stack of the executing thread. Only objects that reside on the stack of the current thread will be destroyed during stack unwinding, and only those functions that are pending on the current stack will be considered for matching catch clauses. Developers need to take all executing threads into consideration when an exception occurs.

A few suggested guidelines for using exceptions correctly in multi-threaded applications are given below:

- All threads in a multithreaded process should not be arbitrarily terminated because of an exception thrown in one thread. A single thread could have acquired a shared resource and unpredictable results can occur if it will not be able to release this resource before the program ends.
- The default version of `terminate()` should be replaced. The replacement `terminate()` should not try to return and continue executing. It is not sufficient to put a `catch(...)` clause at the beginning of each thread. The `terminate()` function may be called for other reasons than un-handled exceptions such as a corrupted stack, or an additional exception being thrown from a destructor.
- Make sure multithreaded applications can shut down all threads gracefully in the event of a fatal exception, or if exception processing is unable to continue. The thread that first detected the exception should be terminated but the other threads in the process should be notified also. This allows them to make an appropriate response such as cleaning up and shutting down gracefully.

4.9 Using Multi-thread Options In Makefiles

Whenever a developer has used multi-threaded processes within an application, that application should use makefiles with multi-threaded process options enabled. Enabling these options can reduce performance and increase memory usage, therefore, they should not be arbitrarily used for all applications. The following instructions should be followed:

1) You must include the file:

```
/ecs/formal/COMMON/make/make.options
```

Make sure that the OODCE, DCE, and RWTOOL (RW_POSIX_THREADS) options/sections are present.

2) In your makefile you must have:

```
CXXFLAGS += $(OODCECXXFLAGS)

LDLIBS += $(OODCECXXFLAGS)
```

4.10 The Real World

The behavior of multi-threaded processes is somewhat dependent on the platform on which they are executed. For example, the number of threads permitted to execute concurrently on Solaris is in excess of 10,000 threads, while under IRIX 3500 threads are all right, and under HP-UX 9.x exceeding 600 threads can cause you to run out of memory. Additionally, the HP currently does not seem to switch between threads as often as the SPARC. However, HP claims that with HP-UX 10.x things will improve. As a result of this, the mechanisms employed to construct thread safe applications are also somewhat dependent on the platform.

In general, regardless of the platform, as soon as you start dealing with more than 2 mutexes that will be locked to access a resource, it can become tedious verifying that you always use the mutexes in the proper order.

Also, It is a good practice to always put the thread include files (`Pthread.H` or `pthread.h`) first in your program. On the HP, this is absolutely necessary since they define non-thread safe calls into the thread safe equivalents.

The following list presents many of the known problems on the HP platform:

- `rand_r()` works differently on the HP than on the SPARC. It has different arguments and a different return value.
- file operations can block the entire process instead of just the thread.
- `ofstreams` are not thread safe. When you try to compile, it will complain that `cma_open()` and `cma_close()` are not defined. If you force the compiler to use the old `open()` and `close()` for `ofstreams` you must use mutexes to prevent multiple concurrent calls to `open()`. And the process, not just the thread, may block on the `open()`.
- `rand_r()` works differently on the HP than on the SPARC. It has different arguments and a different return value.
- Calls like `system()` may work strangely on these systems. The `system()call` seems to block the entire process until the system call has returned.
- `fork()` causes serious problems. `fork()` may cause threads to get `EINTR` on any I/O activity. In addition, there is a serious conceptual problem involved. For example, a process may have two threads. One thread is about to write a message to a file on the disk while another thread calls `fork()`. Now, there are two copies of that program running, and thus there are two threads, each in a separate process, neither of which called or are aware of the `fork()`. Both processes are then about to write to disk.
- `lockf()` will grant the same lock to multiple threads in the same process. So, while with multiple processes, locks are exclusive, in a multi-threaded environment threads must be careful to make sure that no other threads already have a `lockf()` on a given file.
- While functions like `cout << msg` and `cerr << msg` are thread safe, and will not crash, messages may become interleaved.

4.11 Suggestions And Reminders

You will make mistakes in using mutexes -- check your code carefully. Remember mutexes do not protect data members unless you consistently use them correctly. It is extremely important to test your code as well as you possibly can, and then test it some more. When testing your code, try to run it in an environment with hundreds of concurrent threads. This will help to insure that the code is well written and will behave well in a multi-threaded environment.

Finally, all the bugs in DCE/OODCE have not be uncovered. You should always be aware that you may stumble across more. It is important that your discoveries are brought forward and made known to the rest of the team using ccMail bulletin boards and the release architect.

4.12 Other Areas Of Interest

This guide provides only brief discussions on the issues concerning programming with multi-threaded processes. Some additional suggested topics for further examination to enhance the readers understanding and use of threads include:

- Thread priorities; specifying some threads to have higher priority than others
- Scheduling policy; changing how threads are scheduled on the available processors

Thread signaling; having threads wait for signals (not UNIX signals) and send signals to awaken threads.

5. Conclusions

This is a “living” document that will evolve as new COTS products are brought into the environment, ongoing trials and investigations are concluded, and new information becomes available. The information contained within this document is complete as far as the POC information were complete. Whenever possible references are provided to other books, manuals, and resources to obtain more in depth information. This document is intended to provide supporting engineering analysis to form the basis from which risk mitigation plans can be built, not as the definitive answer to problems presented.

The results of the COTS assessments reveal that the risk of these products causing applications to crash due to multi-threaded processing is not widespread, and is isolated to only a few products. For most of the products at risk, possible work around solutions are either being evaluated or investigated.

The high risk COTS software libraries are the Sybase DB-library and the RogueWave DbTools.h++ because of its use of the Sybase client library. Evaluations are currently being conducted to determine if version upgrades to Sybase will mitigate these problems.

The high risk COTS development and test tools are in the areas of the platform compilers and debuggers. The DEC C++ compiler has no multi-threaded process compile options, which presents a significant problem for developers in building any applications needing multi-threaded processes on that platform.

Currently, only the SUN platform provides a multi-threaded process debugger. This debugger is installed in the environment and is undergoing evaluation. None of the other platforms provide a true multi-threaded process debugger. This will impact integration and testing, since multi-threading processing problems that could not be detected and corrected in unit test may find their way into integration and testing.

It is important that procurement personnel, integrators, and developers, at all levels of experience, be cognizant of the ramifications of introducing non-thread safe software into the environment. Applications and libraries are intended to be shared through out the environment and as a result problems can and will be promulgated and escalated.

This page intentionally left blank.

References

- Andrew S. Tanenbaum, “Modern Operating Systems”, Prentice Hall, New Jersey, 1992
- John Shirley, “Guide to Writing DCE Applications”, O’Reilly & Associates, Inc., 1992

This page intentionally left blank.

Abbreviations and Acronyms

API	Application Programming Interface
COTS	Commercial Off The Shelves
DCE	Distributed Computing Environment
ECS	EOSDIS Core System
FOS	Flight Operations Segment
GUI	Graphic User Interface
HP	Hewlett-Packard
IPC	Inter Process Communications
OODCE	Object Oriented DCE
OS	Operating System
POC	Point Of Contact
RFC	Request For Comments
RFP	Request For Procurement
RPC	Remote Procedure Call
SEPG	Software Engineering Process Group
SDSRV	Science Data Server
SGI	Silicon Graphics Incorporated

This page intentionally left blank.